1987-01

A Relational Information Resource Dictionary System

Dolk, Daniel R.; Kirsch, Robert A. II

Edgar H. Sibley
Panel Editor

A relational implementation of IRDS using SQL demonstrates how the flexibility of the relational environment enhances the extensibility of the IRDS while at the same time providing more powerful dictionary capabilities than are typically found in relational systems.

A RELATIONAL INFORMATION RESOURCE DICTIONARY SYSTEM

DANIEL R. DOLK and ROBERT A. KIRSCH II

The problem of controlling and administering organizational information resources is an increasingly complex one for today's MIS manager. The exponential growth of computer technology and the corresponding demand for information have placed a higher priority on the effective management of these resources, but the effort to keep pace with technology often leaves little time for the proper administration of this technology. Consider the resolution of the following typical hypothetical questions:

- How many files and programs will be affected by changing zip codes from five to nine digits?
- To make the transition from a Fortran to an Ada environment, how many programs and lines of code will have to be converted?
- To help plan a distributed processing environment, what departments use which files and programs what percentage of the time?
- What computing equipment (including personal computers) does the organization have, and what is it used for?
- To implement a data-management environment, which files and programs must be converted for database processing?

Although it is difficult to imagine an effective information planning, control, and operations environment without timely access to this kind of information, relatively straightforward questions like these continue to stymie many managers.

Historically, data management has been concerned primarily with the development of effective database management systems (DBMSs) to facilitate data sharing, reduce data redundancy, and provide an integrated environment for data manipulation. An integral part of such a DBMS is the data dictionary (D/D), a catalog of information about the logical and physical aspects of the data environment. In sharp contrast to the operational databases developed within the DBMS environment, the D/D is concerned with the *description* of the operational databases rather than the actual data values contained within those databases.

In recent years, the scope of the D/D has been expanded to include a wider range of information resources: that is, to include basically any information entity—a program, user, hardware, or decision model. This enhanced notion of a D/D is referred to here as an information resource dictionary system (IRDS).

It is estimated that the federal government can realize as much as $120 million in benefits by the early 1990s from the use of a standard IRDS [3]. To help realize these savings, the National Bureau of Standards has developed specifications for an IRDS that will form the basis for a Federal Information Processing Standard IRDS (FIPS IRDS) [3–6]. These specifications include many of the functions available in existing commercial D/D systems [1, 14], but

This research was supported by grants from the U.S. Army Military Personnel Center and the Naval Ocean Systems Center.

also provide flexibility for tailoring the IRDS to specific information administration requirements.

The development of these specifications should significantly facilitate the corresponding development of IRDSs for relational DBMS (RDBMS) environments. Typically, RDBMSs are strongly oriented toward implementing operational databases and do not adequately support the administrative aspects of information management [14]; they tend to be performance rather than administration oriented. Moreover, RDBMS dictionaries are generally concerned only with their own data resource environment, not the overall information environment, and are usually hard-wired so that data administrators cannot modify them for their own requirements.

In this article, we describe a relational model of a passive IRDS (i.e., a "stand-alone" IRDS) that is consistent with a subset of the FIPS specifications and can easily be implemented and used with existing RDBMS products. This model is then enhanced by adding another metalayer, which allows the IRDS to become self-descriptive. Finally, we show how the addition of this metalayer facilitates IRDS extensibility, and the migration to an active IRDS that interacts with other system components, by demonstrating an actual implementation of the enhanced model using the ORACLE RDBMS. We argue that our relational implementation of the logical kernel of the FIPS IRDS specifications provides the following benefits:

- a more comprehensive dictionary capability than most commercially available RDBMSs offer,
- a flexible prototype for developing organizational dictionaries that is easy to implement and can be adapted to a wide range of applications and environments,
- compatibility with an emerging federal standard,
- a basic relational implementation of the entity-relationship model [8], and
- a foundation for considering more powerful semantic models for metadata management.

The first two introductory sections of the paper offer brief overviews of dictionary terminology and the FIPS IRDS specifications, respectively. For a more complete discussion of dictionary concepts and the potential applications of dictionary systems, see [15] and [16].

INFORMATION RESOURCE DICTIONARY SYSTEMS: A REVIEW OF THE TERMINOLOGY

An IRDS is a logically centralized repository of data about all relevant information resources within an organization. Since the data within an IRDS describe other data, they are often referred to as *metadata*.

The *dictionary* component of an IRDS describes what information resources exist, what they mean, and what their logical structures are. An IRDS may also have a *directory* component describing where information resources are located and how they are accessed. For the data resource, dictionary and directory are roughly analogous to logical and physical descriptions, respectively. A dictionary description of an employee file might contain the fields within the file, sources for the data, and data integrity constraints; whereas the directory description would contain data on the machine, operating system, and file structure under which the file is stored.

Dictionaries are classified as either *passive* or *active* in nature. A passive IRDS is one in which no process or system component depends on the IRDS for its metadata, whereas an active IRDS generates metadata for one or more processes and is the sole source for those metadata. A common example of an active IRDS would be where a DBMS consults an IRDS for all information concerning the data entities within a particular operational database. Passive systems are used primarily for documentation purposes and require separate transactions for registering metadata. Active systems are much more powerful in implementing control mechanisms, but extract a performance penalty since all transactions must go through the IRDS. A common implementation strategy is to build a passive system and then extend it to an active system for selected applications; this process of activating the RIRDS is reviewed briefly
on page 59.

IRDSs can also be characterized as either *DBMS dependent* or *freestanding* (*DBMS independent*). A DBMS-dependent IRDS uses an existing DBMS to implement the description, manipulation, and control of its metadata, and therefore can avail itself of the underlying query processor, security, backup/recovery, and other features. A freestanding IRDS, on the other hand, must supply those capabilities internally. The advantage of the freestanding IRDS is that it does not require a specific DBMS environment and is therefore more versatile for multi-DBMS and distributed database applications.

FEDERAL INFORMATION PROCESSING STANDARDS FOR INFORMATION RESOURCE DICTIONARY SYSTEMS (FIPS IRDS)

The Institute for Computer Sciences and Technology of the National Bureau of Standards has developed specifications for an IRDS that will form the basis for a FIPS. These specifications have been accepted by the American National Standards Committee X3H4 as the basis for its draft proposed American National Standard IRDS [3–6].

FIPS DDS Design Objectives

The first of three major design objectives defined for the IRDS was that it contain the major features and capabilities found in existing dictionary systems. All major vendors of dictionary systems were asked to review and make recommendations on various drafts of the standards. These suggestions were eventually incorporated into a *core* dictionary system consisting of a *system-standard schema*, three modules (entity level security, application program interface, and data model support), and two user interfaces (panel and command language).

The second design objective was to make the IRDS as flexible as possible, in recognition of the fact that no single standard will be able to satisfy the unique requirements of all users. Thus, the system-standard schema represents a consensus concerning the entities, attributes, and relationships that should be available in an IRDS. However, this does not preclude the addition of other entities, attributes, and relationships by individual users or vendors. To contribute further to this flexibility, the FIPS IRDS does not require both interfaces or any of the modules to be in the IRDS, only that they be independent of one another, which allows users to choose which options they wish in their IRDS.

The third major objective was that the IRDS support portability of skills and a wide range of user environments. This resulted in specifications for a menu-driven panel interface for the inexperienced user and a command language interface for the more experienced user. An implementation of the IRDS standard is considered complete if either of the interfaces is implemented.

Features of the IRDS

The FIPS IRDS specifications include a collection of entity-types, attribute-types, and relationship-types comprising the *core system-standard schema.* The core is expected to be a part of every implementation conforming to the standard, and can be extended, if necessary, by adding additional schema descriptors as required. Below is a brief discussion of the core as it relates to the relational IRDS described on page 52. For a more detailed account of the core and the IRDS specifications, see [11].

The IRDS architecture is based on the entity-relationship (E-R) model [8] and is comprised of the Information Resource Dictionary (IRD) and the IRD schema. The IRD consists of entities, attributes, and relationships that are instances of the corresponding IRD schema entity-types, relationship-types, and attribute-types. The IRD schema, in turn, consists of instances of metaentities, metarelationships, and metaattributes at the IRD schema description level. Thus, the IRD may contain the data PERSONNEL_RECORD, which is an instance of the entity-type RECORD in the IRD schema, which, in turn, is an instance of the metaentity ENTITY_TYPE at the schema description level (Figure 1).

The structure of the IRDS is similar to a semantic network where entities are the nodes, and relationships are the arcs that connect nodes. All relationships are binary, and entities may be related to themselves. Attributes may be associated with either entities or relationships. Another key characteristic of the IRDS is that it is strongly typed: Each instance of an entity, attribute, or relationship corresponds with an instance of an entity-type, attribute-type, or relationship-type, respectively. The entity-, attribute-, and relationship-types comprising the core are shown in Figure 2.

Entity-Types. The IRD schema contains 12 entity-types that are categorized as either data, process, or external. FILE, RECORD, ELEMENT, and DOCUMENT are the major data entity-types. BIT-STRING, CHARACTER-STRING, FIXED-POINT, and FLOAT are also data representation entity-types used by the REPRESENTED-AS relationship to describe characteristics of ELEMENTS. SYSTEM, PROGRAM, and MODULE comprise the system entity-types, and USER is the sole external entity-type.

IRD schema description layer	Entity-type	Relationship-type	Attribute-type
IRD schema layer	ELEMENT, RECORD, etc.	RECORD-CONTAINS-ELEMENT	DATE-ADDED, LENGTH, LOCATION, etc.
IRD data layer	Soc-Sec-No, Empl-Record, etc.	Empl-Record-CONTAINS-Soc-Sec-No	23Nov85, 12 (Char) Bldg A-Room 3
Operational data	555-23-6666 (Employee record for Kirk)	Empl-Record for Kirk-CONTAINS-(555-23-6666)	(Attributes do not appear as instances in operational databases)

FIGURE 1. IRDS Architecture

Entity-types

SYSTEM	FILE	BIT-STRING
PROGRAM	RECORD	CHARACTER-STRING
MODULE	ELEMENT	FIXED-POINT
USER	DOCUMENT	FLOAT

Attribute-types

Entity related:

ACCESS-NAME	DURATION-VALUE
ADDED-BY	HIGH-OF-RANGE
ALLOWABLE-VALUE	LAST-MODIFICATION-DATE
ALTERNATE-NAME	LAST-MODIFIED-BY
CLASSIFICATION	LOCATION
CODE-LIST-LOCATION	LOW-OF-RANGE
COMMENTS	NUMBER-OF-LINES-OF-CODE
DATA-CLASS	NUMBER-OF-MODIFICATIONS
DATE-ADDED	NUMBER-OF-RECORDS
DESCRIPTION	RECORD-CATEGORY
DESCRIPTIVE-NAME	SECURITY
DOCUMENT-CATEGORY	SYSTEM
DURATION-TYPE	

Relationship related:

ACCESS-METHOD	FREQUENCY
RELATIVE-POSITION	

Relationship-types

CONTAINS	GOES-TO
PROCESSES	CALLS
RESPONSIBLE-FOR	DERIVED-FROM
RUNS	REPRESENTED-AS

FIGURE 2. Core System-Standard Schema Types

Attribute-Types. Attribute-types in the core system-standard schema were selected as those most likely to be needed by organizations to describe their information environments: They are intended to provide audit trail information as well as general documentation for entities and relationships.

The core allows several distinct names to be associated with an entity: ACCESS-NAME, DESCRIPTIVE-NAME, and ALTERNATE-NAME. The ACCESS-NAME is the primary name with which the user will interact; it should be short for ease of use and must be unique throughout the IRDS. The DESCRIPTIVE-NAME allows a more detailed and meaningful name to be assigned to an entity so that the brevity of the ACCESS-NAME is not a restriction; it must also be unique throughout the IRDS. The ALTERNATE-NAME provides a "synonym" or "alias" capability so that different names can be assigned to the same entity.

Not all attribute-types shown in Figure 2 apply to all entity-types. NUMBER-OF-LINES-OF-CODE, for example, applies only to the PROGRAM and MODULE entity-types (see [3] for more details). The following attribute-types, however, pertain to all but the data representation entity-types: ACCESS-NAME, ADDED-BY, CLASSIFICATION, COMMENTS, DATE-ADDED, DESCRIPTION, DESCRIPTIVE-NAME, ALTERNATE-NAME, LAST-MODIFICATION-DATE, LAST-MODIFIED-BY, NUMBER-OF-MODIFICATIONS and SECURITY.

Some of the attribute-types are also multiple attributes in that they are in a many-to-one relationship with entity-types. ALTERNATE-NAME, for example, may have several different values for any particular entity-type with which it is associated. Thus, the entity 'ZIP_CODE' may have ALTERNATE-NAME attribute values 'ZCODE', 'ZIP', and 'ZIPCODE'. Multiple attribute-types in Figure 2 are ALLOWABLE-RANGE, ALLOWABLE-VALUE, CLASSIFICATION, CODE-LIST-LOCATION, and LOCATION.

Relationship-Types. The relationship-types provided by the IRD schema are designed to capture the important associations between entities that apply in an information resource environment. All relationship-types in the core system are binary and are named self-descriptively according to the entity-types that participate in them (e.g., SYSTEM-CONTAINS-PROGRAM).

Restrictions concerning which entity-types can participate in which relationship-types (Figure 3, next page) serve to define the integrity constraints that will be applied to the underlying entity and relationship metadata. Relationship-types may have associated attribute-types as well: For example, the IRD schema allows the attribute-type ACCESS-METHOD for the relationship-types SYSTEM-PROCESSES-FILE, PROGRAM-PROCESSES-FILE, and MODULE-PROCESSES-FILE.

Functions and Processes. The core IRDS must support the description, manipulation, and control of entity-, attribute-, and relationship-types as well as particular instances of the same. Schema maintenance and output involve the ability to describe entity- and relationship-types and display information about the types existing in the IRDS. In other words, the IRDS must be *self-descriptive.*

IRDS population, maintenance, and output refer to the creation, manipulation, and display of actual entity and relationship instances involving the data about the information resources themselves (as opposed to the logical description of the resources). Thus, schema maintenance is involved in describing the entity-types FILE and PROGRAM and the relationship-type PROGRAM-PROCESSES-FILE, whereas IRDS maintenance would be involved

```
CONTAINS(system,system)
CONTAINS(system,program)
CONTAINS(system,module)
CONTAINS(program,program)
CONTAINS(program,module)
CONTAINS(module,module)
CONTAINS(file,file)
CONTAINS(file,document)
CONTAINS(file,record)
CONTAINS(file,element)
CONTAINS(document,document)
CONTAINS(document,record)
CONTAINS(document,element)
CONTAINS(record,record)
CONTAINS(record,element)
CONTAINS(element,element)

RESP_FOR(user,file)
RESP_FOR(user,document)
RESP_FOR(user,record)
RESP_FOR(user,element)
RESP_FOR(user,system)
RESP_FOR(user,program)
RESP_FOR(user,module)

RUNS(user,system)
RUNS(user,program)
RUNS(user,module)

CALLS(program,program)
CALLS(program,module)
CALLS(module,module)

PROCESSES(system,file)
PROCESSES(system,document)
PROCESSES(system,record)
PROCESSES(system,element)
PROCESSES(program,file)
PROCESSES(program,document)
PROCESSES(program,record)
PROCESSES(program,element)
PROCESSES(module,file)
PROCESSES(module,document)
PROCESSES(module,record)
PROCESSES(module,element)
PROCESSES(user,file)
PROCESSES(user,document)
PROCESSES(user,record)
PROCESSES(user,element)

DERIVED_FROM(document,file)
DERIVED_FROM(document,document)
DERIVED_FROM(document,record)
DERIVED_FROM(element,file)
DERIVED_FROM(element,document)
DERIVED_FROM(element,record)
DERIVED_FROM(element,element)
DERIVED_FROM(file,document)
DERIVED_FROM(file,file)
DERIVED_FROM(record,document)
DERIVED_FROM(record,file)
DERIVED_FROM(record,record)

GOES_TO(system,system)
GOES_TO(program,program)
GOES_TO(module,module)
```

The FIPS IRDS expresses relationships as `ENTITYTYPE-RELSHIP-ENTITYTYPE` (e.g., `SYSTEM-CONTAINS-SYSTEM`). Instead, we represent relationships as `RELSHIP(entitytype,entitytype)`.

FIGURE 3. IRDS Relationships

in entering data about the PROGRAM entity EMPL_UPDATE, the FILE entity EMPL_PROFILE, and the relationship PROCESSES(EMPL_UPDATE, EMPL_PROFILE).

An important IRDS output is the *impact-of-change* report, which lists all entities affected by a change to one or more other entities. Other output specifications involve specific report formats for displaying entity and relationship information.

IRDS control facilities include the following:

- *Versioning*—allows description of multiple versions of the same entity (e.g., a program).
- *Life-cycle phase*—allows each entity to be assigned to a life-cycle phase and provides integrity rules for moving from one phase to another.
- *Quality indicators*—facilitate definition of quality indicators that can be assigned to entities.
- *Views*—allow different users to have different views of the dictionary.
- *Security*—regulates authorization and access to the contents of the dictionary.

RELATIONAL IRDS MODEL

Most relational DBMSs provide a relatively narrow range of dictionary capabilities [14], primarily because RDBMSs are concerned mainly with data resources and do not accommodate other information resources such as hardware, software, and decision-making models. Secondly, concentration on database performance tends to dominate other considerations. The technological problems of building an efficient and effective RDBMS are complex and still being investigated (e.g., efficient query optimization is still a key issue in the performance of relational systems, especially with very large databases). Finally, RDBMS dictionaries are hard-wired into their respective systems and cannot be modified.

Since each RDBMS vendor has its own idea about

what should be included in a dictionary system, there is little compatibility or uniformity in the features these systems provide. Furthermore, there is often limited flexibility in changing or adding to the entity-, attribute-, and relationship-types forming the foundation of the dictionary. However, building an organizational IRDS is a complex task, and one can reasonably expect that, like any large information system, it will change dramatically during its development life cycle.

The FIPS IRDS specifications provide a standard core from which dictionary systems can be implemented. The power and flexibility of RDBMSs provide many built-in features (e.g., query languages, report generators, security mechanisms, and views) to facilitate this implementation. In this section we describe a relational model of an IRDS that is compatible with a major portion of the FIPS IRDS core specifications and can easily be implemented and used in existing RDBMS environments to extend information administration.

Design Objectives

The design objectives incorporated in the relational IRDS (RIRDS) model and the subsequent implementation are as follows:

- *Passivity and RDBMS dependency.* The RIRDS should be freestanding. (Approaches to activating the RIRDS are discussed on p. 59.)
- *Compatibility with a significant subset of the FIPS IRDS specifications.* This means that most of the entity-, attribute-, and relationship-types specified in the core system-standard schema are included.
- *Ease of implementation and use.* The RIRDS model should be easy to implement in an RDBMS environment, and all operations on the RIRDS should be accomplished via the RDBMS query language.
- *Extensibility.* To incorporate nonstandard entity-, attribute-, and relationship-types, and other features in order to accommodate user-specific requirements. The flexibility of the relational model (i.e., the ability to change logical descriptions independently of the underlying physical storage) provides powerful extensibility options.
- *Self-descriptiveness.* The RIRDS should be able to describe the entity-, attribute-, and relationship-types comprising it. The administrator should be able, via a simple query, to determine which entity-types are involved in the CONTAINS relationship-type.

RIRDS Model

Since the FIPS IRDS is based directly on the E-R model, the relational model for the RIRDS must first and foremost capture this essence. This is done straightforwardly with two relations, ENTITY and RELSHIP, as shown in Figure 4. (All relations are in uppercase, and attributes of a relation in lowercase. Keys are underlined.) Note that values for ENTITY.etype and RELSHIP.rtype must come from the domain of acceptable IRDS entity-types and relationship-types, respectively, as shown in Figure 2; and the nonkey attributes associated with ENTITY and RELSHIP include all appropriate core attribute-types (as shown in Figure 2), with the few exceptions discussed below (i.e., primarily those designated as multiple attributes). Self-evident abbreviations are used for the FIPS attribute names (e.g., aname for ACCESS-NAME, and dname for DESCRIPTIVE-NAME).

```
ENTITY(ename,etype,dname,added-by,
       date-added,mod-by,last-mod,nmods,
       dur-value,dur-type,comments,descr,
       security,lang,lines-code,nrecs,
       rec-cat,data-class,doc-cat)
RELSHIP(rtype,e1name,e1type,e2name,e2type,
        access-method,frequency,rel_pos)
```

FIGURE 4. Basic Relational Representation of the IRDS Entity-Relationship Model

With this E-R model as a base, the RIRDS model (Figure 5, next page) can be conveniently superimposed using the view feature provided by RDBMSs. Notice, for example, that the SYSTEM entity-type is simply a subset of the ENTITY relation for the case etype='SYSTEM'. Similarly, the SYSTEM-PROCESSES-FILE relationship-type is simply a subset of the RELSHIP relation for the case rtype='PROCESSES' & e1type='SYSTEM' & e2type='FILE'. (Further discussion of this procedure occurs on p. 55 under "Implementation.")

Compatibility with FIPS IRDS. The primary focus of this RIRDS implementation has been to capture the entities, attributes, relationships, and meta-equivalents comprising the FIPS IRDS core system-standard schema. Implementation of IRDS control facilities such as versioning and quality indicators has not been attempted.

The entity-types BIT-STRING, CHARACTER-STRING, FIXED-POINT, and FLOAT, which are included in the core system-standard schema, have been left out of the RIRDS, as has the REPRESENTED-AS relationship-type, because these types are concerned with the physical representation of data (primarily ELEMENT entities) and thus comprise the directory portion of the IRDS. Although it is appropriate to provide this in an IRDS, we feel that the FIPS approach unnecessarily precludes representing

Entities and attributes

SYSTEM(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,dur-value,
dur-type,comments,descr,security)

PROGRAM(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,dur-value,
dur-type,lang,lines-code,comments,
descr,security)

MODULE(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,dur-value,
dur-type,lines-code,comments,descr,
security)

FILE(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,nrecs,
comments,descr,security)

RECORD(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,rec-cat,
comments,descr,security)

ELEMENT(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,data-class,
low-range,high-range,comments,
descr,security)

DOCUMENT(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,doc-cat,
comments,descr,security)

USER(aname,dname,added-by,date-added,
mod-by,last-mod,nmods,comments,descr,
location,security)

Relationships

All relationships have the same attributes and keys:

REL(e1name,e1type,e2name,e2type)

where e1name,e2name are the entity instances

e1type,e2type are the entity-types of which e1name,e2name are instances, respectively

REL is any of the relationships CONTAINS, PROCESSES, RUNS, RESP_FOR, CALLS, GOES_TO, DERIVED_FROM, ALIAS, and KWIC

Integrity constraints

See Figure 3

FIGURE 5. Relational IRDS (RIRDS)

many realistic situations: for example, the element entity 'SOCIAL_SECURITY_NUMBER', which may appear as a FIXED-POINT element in one file and a CHARACTER-STRING in another. Because of the binary nature of relationships, this information cannot be recorded in the IRDS; the equivalent of a tertiary relationship such as ELEMENT-REPRESENTED-AS-DATA-TYPE-IN-FILE is necessary to capture it. Therefore, we recommend embedding this information in the FILE-CONTAINS-ELEMENT relationship-type in the form of an attribute-type such as FORMAT.

The core attribute-types that have been omitted from the RIRDS are primarily those designated as multiple attributes in Figure 2. The multiple attribute designation results in relations with repeating groups, which violates first normal form. Therefore, to include those attributes in the model, new relations have to be defined. In cases where these attributes are vital, the relations are included (e.g., ALIAS and KWIC corresponding to the attribute-types ALTERNATE-NAME and CLASSIFICATION, respectively). Otherwise, as with ALLOWABLE-RANGE, CODE-LIST-LOCATION, and LOCATION, the attributes are omitted from the model to streamline the presentation.

Two other interesting features are provided by the RIRDS model: the synonym capability represented by the ALIAS relation, which allows multiple names to be assigned to the same entity—a common occurrence in information processing environments; and the synonym feature, which allows these names to be identified and eventually standardized.

Finally, a key-word-in-context (KWIC) feature is provided by the KWIC relation. Using this feature, entities can be classified according to user-chosen categories facilitating queries such as "List all entities associated with PERSONNEL."

Extensible Features. The extensibility of this version of the RIRDS derives directly from the flexibility of the RDBMS environment, where entity-, attribute-, and relationship-types can be added dynamically as needed by simply adding the appropriate relations or attributes. To add the INPUT_TO relationship-type to complement DERIVED_FROM and provide for information flow descriptions, the information administrator would simply define the INPUT_TO relation with the appropriate attributes in the host RDBMS. More examples are provided in the implementation discussion on the facing page.

Self-Description. In its present form, the RIRDS model accommodates the IRD and the IRD schema. However, it is limited in that it is not possible for the information administrator to know what the model is from the RIRDS itself: That is, there is no way to discover what the entity-types are in the model or which entity-types participate in which relationship-types without resorting to the host RDBMS's dictionary capabilities. Since the RIRDS is intended as a significant extension to current RDBMS dictionaries, it is essential that the RIRDS be able to describe itself.

This is equivalent to implementing the IRD

schema description level given in Figure 1 as an integral part of the RIRDS. This is done by creating the additional "meta" relations ENT_TYPE, ATT_TYPE, and REL_TYPE, as shown in Figure 6; the additional metalayer facilitates description of the IRDS model. Instances of ENT_TYPE are the entity-types supported by the IRDS (e.g., SYSTEM, FILE, USER), and are implemented as views in the RIRDS at the IRD schema level.

Metaentities, -attributes, and -relationships

```
ENT_TYPE(aname,dname,added-by,date-added,
         mod-by,last-mod,nmods,comments,
         descr,security)

ATT_TYPE(aname,dname,added-by,date-added,
         mod-by,last-mod,nmods,comments,
         descr,security)

REL_TYPE(aname,dname,added-by,date-added,
         mod-by,last-mod,nmods,comments,
         descr,security)
```

Integrity Constraints

```
CONTAINS(ent_type,att_type)
CONTAINS(rel_type,ent_type)
```

FIGURE 6. RIRDS Schema Description

The relational prototype we propose can also accommodate the integrity constraints shown in Figure 3. For example, to show that PROGRAM-PROCESSES-FILE is a valid relationship between entity-types, we simply add the tuple PROCESSES ('program','ent_type','file','ent_type') to the RIRDS. To demonstrate that the attribute DATE_ADDED is part of the SYSTEM entity, we add the tuple CONTAINS('system','ent_type', 'date_added','att_type'). In this way, we effectively represent the logical structure of the IRDS, which can then be queried like any other data. This self-descriptive capability means that

- the integrity of the IRDS can be determined and enforced via simple queries;
- extensibility is enhanced because the information administrator can now add new entity-types as well as specify which relationship-types they can participate in;
- activation of the dictionary is facilitated.

The metaentity level information describes the logical structure of the IRD schema level for the instances that will be stored at the IRD level.

Implementation

The RIRDS is intended for implementation on any RDBMS and subsequent manipulation, as with any other database, using standard DBMS capabilities (primarily the query language). To demonstrate the features of the RIRDS, we present an implementation using the ORACLE RDBMS on a VAX 780 operating under VMS. ORACLE was chosen primarily because it supports the SQL data manipulation language, which is under consideration as a federal standard [2]. No claims are made for the relative superiority of either ORACLE or SQL vis-à-vis other systems. The reader is assumed to have some familiarity with SQL syntax and the notion of subqueries (see [7] and [9] for more details on SQL).

Creation of ORACLE Tables. The RIRDS is implemented very straightforwardly in ORACLE by creating the relations and views given in Figures 4, 5, and 6 using the SQL CREATE TABLE and CREATE VIEW commands. (See Figure 7, on the next page, for examples of this process.) Although several relation names are reserved words in ORACLE and therefore unavailable for use (e.g., FILE, USER, and CONTAINS), for purposes of this discussion the original entity-type names will be used.

Entering Schema Description Information. Once the appropriate relations have been created, the schema information is entered in the RIRDS, establishing the self-descriptive capability of the system. All operations involving entity-, attribute-, and relationship-types should be handled by the information resource administrator. Schema data are entered using the SQL INSERT INTO tablename VALUES command as follows:

1. Enter all entity-types (including ENT_TYPE) as tuples in the ENT_TYPE relation (Figure 8a, p. 57).
2. Enter all attribute-types as tuples in the ATT_TYPE relation and the entity-types or relationship-types to which they belong in the CONTAINS view (Figure 8b, p. 57).
3. Enter all relationship-types as tuples in the REL_TYPE relation (Figure 8c, p. 57).
4. Enter all constraints involving which entity-types can participate in which relationship-types, as defined in Figure 3 (Figure 8d, p. 57).

With the schema information entered, the logical structure of the RIRDS is self-contained and can be manipulated just like any other data. This gives the information administrator a high degree of flexibility and extensibility, which is especially valuable for developing dictionary system prototypes. Figure 9 (p. 57) presents several queries for retrieving information about the logical structure of the IRDS.

(a) Create ENTITY and RELSHIP tables.

```
CREATE TABLE ENTITY
(ENAME        CHAR(15)    NOT NULL,
 ETYPE        CHAR(8)     NOT NULL,
 DNAME        CHAR(30),
 ADDED_BY     CHAR(15)    NOT NULL,
 DATE_ADDED   DATE        NOT NULL,
   :
   :
 DATA_CLASS   CHAR(8)
 DOC_CAT      CHAR(8));

CREATE TABLE RELSHIP
(RTYPE    CHAR(12)  NOT NULL,
 E1NAME   CHAR(15)  NOT NULL,
 E1TYPE   CHAR(8)   NOT NULL,
 E2NAME   CHAR(15)  NOT NULL,
 E2TYPE   CHAR(8)   NOT NULL,
 ACC_METHOD  CHAR(10),
 FREQUENCY   CHAR(10),
 REL_POS     NUMBER(5));
```

(b) Create entity and general relationship views.

```
CREATE VIEW PROGRAM AS
(SELECT  ANAME, DNAME, ADDED_BY, DATE_ADDED, MOD_BY, LAST_MOD, NMODS, DUR_VALUE,
         DUR_TYPE, LANG, LINES_CODE, COMMENTS, DESCR, SECURITY
 FROM    ENTITY
 WHERE   ETYPE='PROGRAM');

CREATE VIEW PROCESSES AS
(SELECT  E1NAME, E1TYPE, E2NAME, E2TYPE
 FROM    RELSHIP
 WHERE   RTYPE='PROCESSES');
```

(c) Create specific relationship views.

```
CREATE VIEW PROGRAM-PROCESSES-FILE AS
(SELECT  E1NAME, E2NAME, ACCESS_METHOD
 FROM    RELSHIP
 WHERE   RTYPE='PROCESSES' AND  E1TYPE='PROGRAM' AND
                                E2TYPE='FILE');
```

FIGURE 7. Creating RIRDS Tables and Views in ORACLE

Entering IRD Metadata. Once the logical structure has been entered, the RIRDS is ready to accept metadata about the information resource environment it will support. This involves inserting values into the various entity views (SYSTEM, FILE, USER, etc.) and the appropriate relationships (PROCESSES, CONTAINS, etc.). SQL can then be used to retrieve the desired resource information (see Figure 10, p. 58).

Extending the RIRDS. The FIPS IRDS specifications provide only a baseline logical structure for an IRDS that can then be extended by information administrators to support the idiosyncrasies of their own environments. To see how this process works with the RIRDS, assume that we want to add the information resource MODEL as the first phase in implementing organizational model management [10]. To add the information resource MODEL, we

1. enter a MODEL tuple with the appropriate values in the ENT_TYPE relation (Figure 11a, p. 59);
2. enter the attributes associated with MODEL in the ATT_TYPE and CONTAINS relations (Figure 11b, p. 59);
3. enter the relationships in which MODEL participates in the appropriate relationship relations (Figure 11c, p. 59);
4. create a MODEL table in the RIRDS using the SQL CREATE VIEW command (Figure 11d, p. 59);
5. enter model metadata (Figure 11e, p. 59).

A similar process is employed for adding attribute-

(a) Entering entity-types in RIRDS

```
INSERT INTO ENT_TYPE VALUES
('program','computer_program','dolk',
 '23Nov85',null,null,null,'Any computer
 program written in a single language',
 'unclass');
```

(b) Entering attribute-types and entity-types to which they belong in RIRDS

```
INSERT INTO ATT_TYPE VALUES
('aname','access_name','dolk','23Nov85',
 null,null,null,'Access name is a
 unique name in the IRDS','unclass');

INSERT INTO CONTAINS VALUES
('program','ent_type','aname',
 'att_type');
```

(c) Entering relationship-types in RIRDS

```
INSERT INTO REL_TYPE VALUES
('contains','ent_type_contains_ent_
 type','kirsch','23Nov85',null,null,
 null,'An entity-type contains some
 entity-type','unclass');
```

(d) Entering integrity constraints in RIRDS

```
INSERT INTO CONTAINS VALUES
('system','ent_type','program',
 'ent_type');

INSERT INTO CONTAINS VALUES
('program','ent_type','module',
 'ent_type');
```

FIGURE 8. Entering RIRDS Schema Description Information

types to existing entity-types. To add the attribute-type PHONE to the USER entity-type, we add the PHONE tuple to the ATT_TYPE relation, add the CONTAINS('user','ent_type','phone', 'att_type') tuple, use the SQL ALTER TABLE command to change the ENTITY relation, and then redefine the USER view to include PHONE.

Attributes are deleted by redefining the view (using DROP VIEW and CREATE VIEW in SQL), leaving off the attribute(s) to be deleted.

Data Integrity

The self-descriptive RIRDS contains the apparatus for verifying its own data integrity. Consider the problem of determining whether any invalid tuples have been entered in the CONTAINS relationship, where an invalid tuple is one that violates one of the constraints listed in Figure 3. For example, CONTAINS('edit_doc','document','edit_sys','system') is invalid since a document cannot contain a system. Any integrity violations in the CONTAINS relationship can be reported via the following SQL command ("!" is equivalent to "NOT"):

(a) "List all attributes associated with PROGRAM and a description of each."

```
SELECT E2NAME,COMMENTS FROM CONTAINS,
  ATT_TYPE
WHERE E1NAME='program' AND E2TYPE=
  'att_type' AND CONTAINS.E2NAME =
  ATT_TYPE.COMMENTS;
```

E2NAME	COMMENTS
aname	Access name is a unique name in the IRDS
dname	Descriptive name
added_by	User who added entity or entity-type
date_added	Date entity or entity-type added
mod_by	User who modified entity or entity-type
date_mod	Date user modified entity or entity-type
dur_value	Duration value
dur_type	Duration type
nmods	Number of modifications
lang	Source language
lines_code	Lines of code
descr	Description
comments	Explanatory comments
security	Security classification

(b) "Which entity_types participate in RESPONSIBLE_FOR?"

```
SELECT E1NAME,E2NAME FROM RESP_FOR
WHERE E1TYPE='ent_type' AND E2TYPE=
  'ent_type';
```

E1NAME	E2NAME
user	system
user	program
user	module
user	document
user	file
user	record
user	element

(c) "Which entity-types does RECORD contain?"

```
SELECT E2NAME FROM CONTAINS
WHERE E1NAME='record' AND E2TYPE=
  'ent_type';
```

E2NAME
record
element

FIGURE 9. SQL Queries for Schema Description Information

```
SELECT * FROM CONTAINS
WHERE E1TYPE != 'ENT TYPE' AND
      E2TYPE != 'ENT_TYPE' AND
      (E1TYPE,E2TYPE) NOT IN
         (SELECT E1NAME,E2NAME FROM
           CONTAINS
          WHERE E1TYPE = 'ENT_TYPE' AND
            E2TYPE = 'ENT_TYPE')
```

(a) "List all files from which the P3MDATA file is derived."

```
SELECT E2NAME FROM DERIVED_FROM,FILE
WHERE E1NAME='p3mdata' AND E1TYPE='file' AND
      DERIVED_FROM.E2NAME=RECORD.ANAME;
```

(b) "All Fortran programs are to be converted to Ada. List all Fortran programs, the lines of code, and the users responsible for each."

```
SELECT E1NAME,E2NAME,LINES_CODE FROM RESP_FOR,PROGRAM
WHERE E1TYPE='user' AND E2TYPE='program' AND
      RESP_FOR.E2NAME=PROGRAM.ANAME AND LANG='fortran'
ORDER BY E1NAME;
```

(c) "Which users are responsible for files containing the element ZIP_CODE or any of its aliases?"

```
SELECT E1NAME,E2NAME FROM RESP_FOR
WHERE E1TYPE='user' AND E2TYPE='file' AND
      E2NAME IN
      (SELECT E1NAME FROM CONTAINS
       WHERE E1TYPE='file' AND E2TYPE='element' AND
            (E2NAME='zip_code' OR E2NAME IN
             (SELECT E2NAME FROM ALIAS
              WHERE E1NAME='zip_code' AND
                    E1TYPE='element')));
```

FIGURE 10. SQL Queries for Retrieving Metadata

This facility demonstrates the power of being able to mix metadata and IRD schema information in the same query. The subquery (the second SELECT clause) identifies the set of all pairs of entity-types in the IRD schema that can legally participate in CONTAINS. The first SELECT clause then identifies any pairs of entities in the metadata appearing in CONTAINS whose entity-types do not fall in that set. All invalid occurrences can subsequently be deleted from the database by simply changing "SELECT *" to "DELETE" in the above query.

Limitations of the RIRDS Implementation

The RIRDS implementation relies on existing ORACLE functionality to perform its data description and manipulation. Although this provides a powerful dictionary capability for the knowledgeable user, there are shortcomings in the areas of data entry and retrieval performance.

Specifically, data entry via SQL is a tedious process since INSERT INTO tablename VALUES must be typed for every tuple. However, nearly all DBMS products, including ORACLE, provide tools for fashioning input screens and forms that significantly facilitate data entry and which could easily be added to the prototype.

A second problem related to data entry is that the inability to represent derived data in the form of rules sometimes necessitates an unreasonable amount of input. Assume, for example, that we enter the file ORDERS containing the record ORDERS_RECORD, which in turn contains a dozen data elements. If we have entered in CONTAINS the tuple showing that ORDERS includes ORDERS_RECORD as well as the dozen tuples showing what data elements ORDERS_RECORD is comprised of, it is still necessary to add another dozen tuples showing that ORDERS includes the data elements. However, if it were possible to represent a rule of the form

$$\text{CONTAINS}(x,y) \;\&\; \text{CONTAINS}(y,z) \rightarrow \text{CONTAINS}(x,z)$$

it would only be necessary to add the tuple CONTAINS(orders,orders_record) to the IRDS; the CONTAINS(file,element) tuples would then be derived by an appropriate query processor at data retrieval time. However, this would require an extension to SQL, which cannot perform this kind of inferencing.

It is appropriate here to discuss the motivation for this particular RIRDS design. Initially, a simpler approach was considered in which the entity- and relationship-types shown in Figure 5 were represented explicitly as tables rather than views. The major problem with this design was that several important queries could not be answered using a single SQL command. Specifically, in executing the impact-of-change query, "If we change entity X, what other entities will be affected?", the explicit design would require exhaustively searching each

(a) Enter MODEL entity-type.

```
INSERT INTO ENT_TYPE VALUES
('model','mathematical_model','kirsch',
 12Dec85',null,null,null,'Any
 mathematical model','unclass');
```

(b) Enter MODEL attributes in ATT_TYPE if they are not already there (we will assume they are) and associate them with MODEL.

```
INSERT INTO CONTAINS VALUES
('model','ent_type','aname','att_type');

INSERT INTO CONTAINS VALUES
('model','ent_type','dname','att_type');

(etc, etc)
```

(c) Enter the relationships in which MODEL participates.

```
INSERT INTO RESP_FOR VALUES
('user','ent_type','model','ent_type');

INSERT INTO DERIVED_FROM VALUES
('document','ent_type','model','ent_type');

INSERT INTO PROCESSES VALUES
('user','ent_type','model','ent_type');

INSERT INTO RUNS VALUES
('program','ent_type','model','ent_type');
```

(d) Create MODEL view.

```
CREATE VIEW MODEL AS
(SELECT ANAME, DNAME, ADDED_BY, DATE_ADDED,
        MOD_BY, DATE_MOD, NMODS,
        MOD_TYPE,
        MOD_METHOD, COMMENTS, DESCR,
        SECURITY
 FROM   ENTITY
 WHERE  ETYPE='MODEL')
```

(e) Enter model metadata.

```
INSERT INTO MODEL VALUES
('p3m', 'personnel_policy_projection',
 'kirsch', '12Dec85', null, null, null,
 'forecast', 'simulation', 'Predict MOS
 manning and reenlistments', 'unclass');
```

FIGURE 11. SQL Commands to Enter New Entity-Type

relationship—using a series of SQL commands—to see which entities interacted with entity X. The current view-oriented approach, on the other hand, can satisfy the impact of change in one command (assume X is the element ZIP_CODE):

```
SELECT   RTYPE,E1NAME,E1TYPE,E2NAME,
         E2TYPE FROM RELSHIP
WHERE   (E2NAME='ZIP_CODE' AND
         E2TYPE='ELEMENT')
   OR   (E1NAME='ZIP_CODE' AND
         E1TYPE='ELEMENT')
```

The view-oriented design is also more flexible in that it effectively implements a self-descriptive E-R model. Any E-R application can be implemented with this methodology by simply specifying the desired attributes when creating the ENTITY and RELSHIP tables, and then proceeding as described under "Implementation" (p. 55).

A disadvantage of the view-oriented approach is a certain loss of efficiency in that any tuple in ENTITY or RELSHIP is likely to have many null fields for those attributes that do not apply to the view the tuple is instantiating. In the IRDS, this is not a serious problem because the entity-types share many common attribute-types. In the general case, however, this may be the exception rather than the rule. Also, some RDBMSs do not allow insertion or modification operations on views. Entering a SYSTEM entity, for example, would require insertion directly into the ENTITY relation, forcing the user to be aware of which attributes required null values. The ORACLE RDBMS allows insertion and modification of views if they are based on only a single, underlying relation. All views in the RIRDS satisfy this stipulation.

Activating the RIRDS

As a passive dictionary system, the RIRDS is not integrated with other system software components that rely on the dictionary for its metadata. Thus, it is primarily a documentation tool. The true value of an IRDS exhibits itself when the IRDS can be included in an information processing loop as an active control and audit mechanism: for example, at the front end of a DBMS, data design, or requirements analysis software system. Establishing active links between the IRDS and operational information systems dramatically increases the utility of an IRDS.

An important benefit of designing the RIRDS to be self-descriptive is that this also facilitates the activation process. The most logical situation for activating an RIRDS is to integrate it with its host RDBMS to gain the following advantages:

- Integrity constraints can be invoked automatically, as described under "Data Integrity" on page 57.
- A simplified version of SQL can be generated to free users from "navigational" details of their databases.
- The IRDS provides a much richer spectrum of information than current relational dictionaries.
- The IRDS is extensible rather than hard-wired.
- The IRDS can be integrated with other system processes thus reinforcing the concept of centralized usage of data and information resources.

A simplified way of integrating the RIRDS is to build an SQL preprocessor (PRE-SQL) that accepts a modified SQL syntax and then generates appropriate SQL commands for execution by the RDBMS. In many cases, PRE-SQL will eliminate the need for the `FROM table` syntax by searching the attribute list in the `SELECT` clause to determine in which tables they appear. It also means that users need not know which attributes are associated with which tables. However, attribute names must be unique for this to work correctly; otherwise, PRE-SQL has to query the user to resolve the ambiguity.

Although a discussion of the concrete details of how an RIRDS can be integrated with its host RDBMS or other system processes is beyond the scope of this article, the value of an active dictionary—coupled with the additional information that an RIRDS provides for a relational environment—makes this a timely topic worthy of further investigation.

Future Research

In the past, dictionary systems have often been viewed as mundane tools used primarily for documentation and other necessary administrative evils. However, as we understand more about data management, database design, and data semantics, the importance of dictionary systems is bound to increase. The establishment of dictionary standards and a relational implementation are one step in this direction. Fruitful areas of further research in this area include the following:

- *Incorporating other semantic models into the IRDS.* The semantic data model (SDM) [12] subsumes the E-R model in terms of semantic expressiveness. The ability of the IRDS to accommodate more robust models like the SDM must be examined.
- *Incorporating time semantics into the IRDS.* The IRDS versioning and life-cycle-phase functions as well as audit trail requirements imply a semantics of time that neither the FIPS specifications nor the relational model adequately support.
- *Extending the dictionary system to a knowledge-based system.* An IRDS is essentially a knowledge-based system about an organization's information resources [13]. Several of the limitations of the RIRDS (see p. 58) suggest that Prolog might be a more appropriate tool for IRDS development. Requirements for implementing the IRDS as an expert system should be explored.

CONCLUSIONS

As the importance of metadata management grows in tandem with the increased complexity of information resource administration and the increased technological maturity of RDBMSs, the specification of IRDS standards will provide a foundation for building metadata support tools. We have surveyed the FIPS IRDS specifications and developed a relational model that conforms to a subset of those specifications. The implemented model is intended to enhance the dictionary capabilities of existing RDBMSs while remaining easy to implement and use, assuming a knowledge of the host RDBMS's data manipulation language. In its most general form, the RIRDS can serve as an implementation of the E-R model. The shortcomings of the relational model as the implementation medium for the IRDS include the inability to express rules easily and the lack of time semantics for capturing the dynamic aspects of an information resource environment.

The FIPS IRDS recognizes that specifications provide only a baseline from which organizations may fashion an IRDS to meet their unique requirements: An IRDS must be extensible. The relational model presented here supports this extensibility as a result of both the inherent flexibility of the relational environment and the explicit representation of meta-level information that makes the IRDS model self-descriptive. These self-descriptive facilities are also critical in converting the IRDS from a passive, primarily documentation-oriented tool, to an active control mechanism.

The development of an organizational IRDS is a complex project. The relational model presented here can perhaps serve as a working prototype from which to develop an IRDS that is FIPS-compatible, extensible, and capable of being activated.

REFERENCES

1. Allen, F.W., Loomis, M.E.S., and Mannino, M.V. The integrated dictionary/directory system. *ACM Comput. Surv. 14*, 2 (June 1982), 245–286. A comprehensive survey of dictionary/directory system features with examples from commercial systems.
2. American National Standards Institute. (Draft proposed) American National Standard database language SQL. ANSI X3H2, American National Standards Institute, New York, 1984. The FIPS specifications for the SQL database language.
3. American National Standards Institute. (Draft proposed) American National Standard information resource dictionary system: Part 1—Core standard. ANSI X3H4, American National Standards Institute, New York, 1985. The FIPS specifications for the core system-standard schema.
4. American National Standards Institute. (Draft proposed) American National Standard information resource dictionary system: Part 2—Entity-level security. ANSI X3H4, American National Standards Institute, New York, 1985. The FIPS specifications for the entity level security module.
5. American National Standards Institute. (Draft proposed) American National Standard information resource dictionary system: Part 3—Application program interface. ANSI X3H4, American National Standards Institute, New York, 1985. The FIPS specifications for the application program interface module.
6. American National Standards Institute. (Draft proposed) American National Standard information resource dictionary system: Part 4—Support of standard data models. ANSI X3H4, American National Standards Institute, New York, 1985. The FIPS specifications for the data model support module.
7. Chamberlin, D.D., et al. SEQUEL 2: A unified approach to data definition, manipulation, and control. *IBM J. Res. Dev. 20*, 6 (Nov. 1976), 560–575. A description of the first version of SQL.

8. Chen, P. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (Mar. 1976), 9–36. The seminal article describing the entity-relationship model.
9. Date, C.J. *An Introduction to Database Systems, Volume 1*. 3rd ed. Addison-Wesley, Reading, Mass., 1982. A classic textbook on database systems including a discussion of SQL.
10. Dolk, D.R., and Konsynski, B.R. Model management in organizations. *Inf. Manage. 9*, 1 (Aug. 1985), 35–47. Introduction of model management as an important component of information resource management.
11. Goldfine, A. The information resource dictionary system. In *Proceedings of the 4th International Entity-Relationship Conference* (Chicago, Ill., Oct. 28–30). IEEE Press, New York, 1985, pp. 114–122. A concise survey of the FIPS IRDS model by one of its creators.
12. Hammer, M., and McLeod, D. Database description with SDM: A semantic database model. *ACM Trans. Database Syst. 6*, 3 (Sept. 1981), 351–386. A detailed description of the semantic data model.
13. Kerschberg, L., Marchand, D., and Sen, A. Information system integration: A metadata management approach. In *Proceedings of the 4th International Conference on Information Systems* (Houston, Tex., Dec.). The Society for Information Management, Chicago, Ill., 1983, pp. 223–239. Discusses the notion of a dictionary as a corporate knowledge base, and argues in favor of a larger role for dictionary systems.
14. Lefkovits, H.C., Sibley, E.H., and Lefkovits, S.L. *Information Resource/Data Dictionary Systems*. Q.E.D. Information Sciences, Wellesley, Mass., 1983. Provides a low-level introduction to dictionary concepts, and surveys eight commercially available systems.
15. Leong-Hong, B.W., and Plagman, B.K. *Data Dictionary/Directory Systems: Administration, Implementation and Usage*. Wiley-Interscience, New York, 1982. The most thorough treatment of dictionaries currently available. Discusses the concepts, applications, selection, implementation, and management of dictionary systems.
16. Navathe, S.B., and Kerschberg, L. Role of dictionaries in information resource management. *Inf. Manage. 10*, 1 (Jan. 1986), 21–46. A comprehensive overview of the application of dictionaries to information requirements analysis/specification and information modeling.

CR Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*relational dictionary; entity-relationship model*; H.2.7 [**Database Management**]: Database Administration—*information resource administration; information resource dictionary system*

General Terms: Design, Standardization

Additional Key Words and Phrases: Database management system, extensibility, ORACLE, query processor, SQL

Received 1/86; accepted 4/86

Authors' Present Addresses: Daniel R. Dolk, Dept. of Administrative Sciences, Naval Postgraduate School, Monterey, CA 93943; Robert A. Kirsch II, Thayer Computer Center, U.S. Military Academy, West Point, NY 10996.